VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROGRAM PRO KOMUNIKACI PŘES SBĚRNICI CCTALK

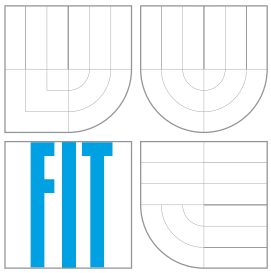
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH DRAHOŠ

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROGRAM PRO KOMUNIKACI PŘES SBĚRNICI CCTALK

PROGRAM FOR COMMUNICATION OVER CCTALK BUS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH DRAHOŠ
AUTHOR

VEDOUCÍ PRÁCE — Dr. Ing. PETR PERINGER
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce řeší problematiku komunikace periferií s řídícím systémem na sběrnici ccTalk. Cílem práce bylo vyvinout program pro řízení této komunikace v prostředí operačního systému Linux. Základem je knihovní rozhraní, které zprostředkovává služby sběrnice ccTalk. Knihovna využívá standardní sériové rozhraní a je napsána s ohledem na zabezpečení přístupu ke sdílenému zdroji.

Abstract

This bachelor thesis deals with the problematics of communications between peripheries and controlling system on ccTalk bus. The goal of this thesis is to develop a program controlling such communication in the environment of Linux operation system. The program is based on a library interface mediating the services of ccTalk bus. The library uses standard serial interface, and it is created with regard on the security of access to a shared source.

Klíčová slova

ccTalk, RS232, Linux, sběrnice, sériová komunikace.

Keywords

ccTalk, RS232, Linux, bus, serial communication.

Citace

Vojtěch Drahoš: Program pro komunikaci přes sběrnici ccTalk, bakalářská práce, Brno, FIT VUT v Brně, 2009

Program pro komunikaci přes sběrnici ccTalk

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Dr. Ing. Petra Peringera

.....
Vojtěch Drahoš
18. května 2009

Poděkování

Rád bych poděkoval Dr. Ing. Petru Peringerovi za vedení bakalářské práce, odbornou pomoc, podnětné nápady a vstřícný přístup při konzultacích. Dále také panu Radkovi Janíčkovi za zapůjčení testovacího hardwaru a pomoc při jeho konstrukci.

Obsah

Kapitola 1

Úvod

Tématem této práce je komunikace po sériových sběrnicích se zaměřením na protokol ccTalk. Důležitou součástí práce je vytvoření softwaru zprostředkujícího sběrnici ccTalk vyšším programovým vrstvám. Do doby zpracování této bakalářské práce neexistovala žádná volně dostupná, funkční knihovní nadstavba nad sběrnicí ccTalk určená pro systémy Linux. K dispozici jsou pouze binární kódy pro hardwarová zařízení, které práci se sběrnicí zprostředkovávají. Toto zadání je zpracováno pro externí firmu Kajot.

Sběrnice ccTalk je díky svému chování vhodná pro využití v široké škále různých zařízení. Byla vyvinuta pro potřeby komunikace mezi zařízeními obsluhujícími finanční hotovost. Průmyslové odvětví, které používá zmiňované technologie, se anglicky nazývá „Coin Industry“, což lze volně přeložit jako průmysl pro zpracování finanční hotovosti. Tento typ průmysl se zaměřuje na výrobu zařízení jako jsou například automaty na jízdenky, parkovací automaty, výherní automaty a podobně.

Úvod práce přibližuje sběrnice jako obecný pojem a důkladně se zaměřuje na možnosti a vlastnosti sběrnice ccTalk. Obsahem kapitoly bude také stručný pohled na některá témata týkající se programování a možností systému Linux.

Třetí kapitola se zabývá možnými řešeními problematiky komunikace se sběrnicí ccTalk. Rozebere klady a zápory jednotlivých přístupů a vybere jeden, který bude předmětem implementace. Také vysvětlí problémy, které bude při implementaci potřeba řešit.

Čtvrtá kapitola pojednává o implementaci zvolené varianty řešení. Popisuje jednotlivé části programu a vysvětluje jak byly implementovány problémy z kapitoly tři.

Kapitola pět popisuje způsob, jakým byl software podroben testům. S tím souvisí i popis programů vytvořených pro demonstraci funkčnosti softwaru.

Závěr shrnuje dosažené výsledky a přínos celé práce. Nastiňuje možnosti rozšiřitelnosti programu jak z pohledu programátora, který chce rozhraní pouze využívat, tak i z pohledu vývojáře rozšiřujícího funkčnost knihovního rozhraní.

Kapitola 2

Problematika sběrnic

Pod pojmem sběrnice si můžeme představit skupinu vodičů, jejímž účelem je propojit jednotlivé části počítačů či jiných zařízení. Pomocí těchto vodičů jednotlivé prvky mezi sebou komunikují. Sběrnice se obvykle skládají z datové, adresové a řídící části.

2.1 Klasifikace sběrnic

Dle způsobu řízení, časování a zapojení rozlišujeme několik základních typů sběrnic. Každý typ sběrnice může mít různé vlastnosti, které se vzájemně doplňují.

Klasifikace sběrnic podle časového signálu

- Synchronní sběrnice jsou takové, kde platnost dat na sběrnici určuje hodinový signál. Jeden z vodičů je určen právě k distribuci hodinového signálu.
- Asynchronní sběrnice jsou takové, kde se začátek datového přenosu určuje podle výskytu předchozí události. Potvrzovací signály slouží k potvrzování komunikace mezi vysláním a přijetím informací. Je pomalejší než synchronní sběrnice. Na rozdíl od synchronní sběrnice může pracovat s různými rychlostmi jednotlivých komponent.

Klasifikace sběrnic podle zapojení

- Nesdílená sběrnice má pro každý typ informace vyhrazenu sadu vodičů.
- Sdílená sběrnice se sdílenými vodiči využívá časový multiplex. Sada je pak postupně využívána různými druhy informací.
- Paralelní je složena z více vodičů určených pro přenos informací. Ve většině případů bývá počet vodičů daný šířkou přenášených informací nebo šířkou potřebnou k zaadresování.
- Sériová sběrnice vysouvá informace na své vodiče postupně po jednotlivých bitech. Je vhodná pro komunikaci na delší vzdálenosti, kde cena kabelu a synchronizační potíže činí paralelní sběrnici nepraktickou. Z konstrukčního hlediska bývají sériové sběrnice implementovány jako sdílené.

Klasifikace sběrnic podle řízení

- Centrálně řízené sběrnice jsou takové, kde o řídícím prvku rozhoduje arbitr. Tento arbitr dle určitých kritérií vybere jednu z komponent a ta má na určený čas sběrnici k dispozici.
- Decentralizované řízení je realizováno bez přítomnosti arbitra. Všechna master zařízení obsahují řídící obvody, které vzájemně spolupracují a hlídají, zda je sběrnice přidělena jen jednomu z nich.
- Multi-master sběrnice jsou takové, kde může více komponent vystupovat jako řídící zařízení. Je potřeba řešit, který z prvků sběrnici ovládá, stejně jako v případě decentralizovaného řízení.
- Single-master sběrnice jsou jednodušší variantou. Pouze jedno zařízení na sběrnici zahajuje a řídí všechnu komunikaci.

2.2 Požadavky na sběrnice v Coin Industry

Paralelní rozhraní sběrnice s sebou přináší výhody jednoduššího a v některých případech rychlejšího přenosu informací. Zjevná nevýhoda je ve větším počtu fyzických spojů, velikosti a krimpování konektorů, problém s odstíněním jednotlivých vodičů a především vyšší cena. Sériové sběrnice redukují počet fyzických spojů na minimum a často nabízejí výhody jako např. jednoduchou rozšiřitelnost nebo různé způsoby autokontroly. Lze je snadno implementovat i v nízkonákladových aplikacích. Sériové rozhraní se také vyplatí v aplikacích s více zařízeními připojených k jednomu řídícímu prvku.

2.3 Sběrnice ccTalk

ccTalk je produktem společnosti Money Controls. Jedná se o sériový protokol určený především pro nízkorychlostní spoje. Protokol vychází z formátování datových rámců stejně jako u protokolu RS232. Díky tomu umožňuje značnou volnost při použití řídících softwaru. Nevyžaduje speciální integrované obvody, vše bývá standardní součástí.

Protokol byl navržen tak, aby mohl propojovat různé druhy periferií. Jelikož se společnost Money Controls specializuje na zařízení obsluhující peněžní hotovosti, je protokol určen hlavně jim. Sběrnice je díky své jednoduchosti použitelná i v jiných průmyslových odvětvích. Základem je třídrátové rozhraní – jeden datový vodič, napájecí a nulový vodič.

Aplikace nad sběrnicí se ve své základní variantě skládá z jedné řídící jednotky a jedné periferie. Pokročilejší aplikace nazývané „Multi-Drop“ sestávají z jedné řídící jednotky a několika periferií s rozdílnými logickými adresami.

Jak uvádí výrobce, protokol byl vybudován zdola nahoru. Nepoužívá od začátku balík nepotřebných vlastností čekajících na příhodnou situaci. Implementuje pouze minimum základních vlastností a v případě potřeby se standard rozšiřuje. ccTalk poskytuje základní seznam příkazů a dává prostor pro nově definované rozšiřující příkazy. Tím se stává ideální pro peněžní průmysl. Protokol nevyžaduje přihlašování ke sběrnici nebo transakční zpracování sekvencí příkazů. Dokáže vykonávat jednoduché úkoly s minimem úsilí.

Velká výhoda využívání formátu RS232 je, že k vytvoření sběrnice mohou posloužit např. staré telefonní dráty s přidanými modemy. Aplikacím nezáleží na vzdálenosti mezi

řídící jednotkou a periferiemi. Délku přenosu lze ošetřit při psaní programu. ccTalk tak může být klíčem k dosažení jednoduchosti a současně bezpečnosti.

Koncem roku 2005 společnost Money Controls představila oficiální testy provozu protokolu ccTalk přes USB. Tyto testy ukazují na rozšíření možností připojení k dnešním počítačům, které sériový port nemají vůbec nebo jej využívají k jinému účelu. Pro připojení sběrnice k USB je potřeba převodník. V dnešní době jsou převodníky poměrně levné a spolehlivé. Po připojení převodníku se systémové ovladače postarají o vznik virtuálního sériového rozhraní, na kterém je sběrnice dostupná. Formátování ccTalk zpráv je identické jak přes USB, tak bez něj. Výhodou připojení „ccTalk over USB“ je, že jednotlivé periferie jsou odděleny USB hubem, a tak nedochází k šíření dat mimo požadovaný fyzický spoj.

Na sběrnici ccTalk může být pouze jeden řídící prvek. V takovém prostředí periferní zařízení nemohou komunikovat mezi sebou, ale musí čekat na výzvu prvku řídícího. V peněžním průmyslu je v dnešní době preferován tento přístup, při kterém má jedno zařízení má veškerou kontrolu nad sběrnicí. Každá komunikace musí být zahájena právě tímto zařízením. Sběrnice je tvořena pouze jedním obousměrným datovým spojem, proto je nutno zajistit, aby nedocházelo ke kolizím.

2.4 Charakteristika sběrnice ccTalk

Sběrnice ccTalk můžeme klasifikovat jako *sdílenou* 2.1, protože využívá stejné vodiče pro veškeré typy komunikace na sběrnici. Jelikož informace vysouvá po jednotlivých bitech, je zároveň sběrnicí *sériovou*. Nevyužívá speciálního vodiče pro přenos hodinového signálu a každý přenos zahajuje a ukončuje speciálním bitem – *asynchronní*. Na sběrnici je povolen pouze jeden řídící prvek, který řídí všechnu komunikaci – *single-master*.

Aby byla sběrnice funkční, musí se při jejím používání dodržovat určité předem dohodnuté standardy komunikace. Takové sadě standardů se říká protokol. Sběrnice ccTalk zavádí stejnojmenný protokol. Ten definuje, jak lze přenášet informace. Zprávu tvoří struktura bajtů. Největším teoretickým softwarovým limitem je limit bajtu a to 255. Ve většině možných situací poskytuje tento limit dostatečný prostor. Oproti možnosti skládání zprávy z různě dlouhých bitových polí je použití bajtu náročnější na paměť a čas, ale značně ulehčuje implementaci.

Protokol používá zásadně osmibitovou strukturu zpráv. Na rozdíl od RS232 nemá ccTalk potřebu devátého budícího nebo adresního bitu. Tato rozdílnost může vyvolávat problémy se špatně nastaveným sériovým portem na PC.

Jelikož je nutné každé zařízení na sběrnici zaadresovat, je jako první bajt zprávy vysílána adresa určeného zařízení. Teoretický maximální počet zařízení připojených k řídícímu prvku je 254. Adresy připojených zařízení nemusí nutně vypovídat o jejich typu. Můžeme tedy připojit více stejných zařízení na rozdílných logických adresách.

Návrh protokolu ccTalk probíhal stylem postupného rozšiřování. Z toho vychází i orientace na typy zpráv. ccTalk místo několika typů zpráv s velkým objemem dat obsahuje více menších a efektivnějších příkazů. Například, jestliže programátor potřebuje zjistit název výrobce zařízení, nemusí jej složitě získávat ze zprávy obsahující spoustu dalších informací, ale pošle jednoduchý příkaz určený přesně pro daný účel.

Protokol dovoluje přenášet informace různými způsoby. Protože délka zprávy není předem daná, je možné přenášet data např. v ASCII znacích. To je užitečné např. u zjištění názvu výrobce, kategorie zařízení nebo zjištění kódu produktu.

Bezpečnost může být zajištěna šifrováním. Některé komponenty mohou zároveň umožňovat spouštění vybraných příkazů jen po předchozím úspěšném zadáním kódu PIN.

Díky své podobnosti se standardem RS232 je možno ccTalk připojit jak k mikro kontrolérům tak i k sériovému portu na PC. Postačí jednoduchý převodník na základě čipu MAX232 a jeho variacích. Komunikace probíhá s rychlostí 9600 Baudů. Z toho vyplývá, že jeden bajt potřebuje k přenosu přibližně 1.04 ms. Aplikacím stačí kontrolovat nová data každou 1 ms. To zaručí, že se nebudou data ztrácet ani v případě levných mikrokontrolérů s malým vyrovnávacím bufferem.

Licence ccTalk

ccTalk je otevřený standard. Slovo „ccTalk“ bylo registrováno jako Evropská obchodní značka a může být použito jako označení na popiskách a v manuálu produktu, který je připraven používat tuto sběrnici. Výrobce ccTalku Money Controls preferují u ostatních výrobců používání označení ccTalk®.

2.4.1 Srovnání s dalšími sériovými protokoly

Hlavní vlastnosti sběrnice ccTalk jsou uvedeny v následující tabulce. Oproti většině ostatních sériových sběrnic se ccTalk odlišuje oboustraným datovým vodičem. Využívá pro přenos informaci nízkorychlostní pásmo 9600 Baudů, což snižuje indukci na vodiči a tím umožňuje fyzicky delší spoje. Protokol k přenosu osmi bitů informací potřebuje napřed na sběrnici vysunout jeden „start bit“. Po datových bitech následuje jeden „stop bit“.

V následující tabulce 2.1 je srovnání parametrů některých sériových protokolů s protokolem ccTalk.

Protokol	**Architektura**	**Rychlost**	**Kontrola**	**Formát**
ccTalk	Obousměrná	9600	8 bit	$1, 8, 1$ bez parity
HI[2]	Obousměrná	9600	8 bit	$1, 8, 1$ bez parity
MDB	TX + RX	9600	8 bit	$1, 8, 1, 1$ adresní bit
BACTA Dataport	TX + RX	1200 nebo 9600	8 bit	$1, 8, 1, 1$ lichá parita
USB	D+ D-	$12\,M$	5 bit CRC	
USB2	D+ D-	$480\,M$	-	-
Bluetooth	rádiový signál	$720\,K$	-	-

Tabulka 2.1: Srovnání základních parametrů sériových sběrnic.

2.4.2 Protokol ccTalk - časování

Časování přenosu datových bitů odpovídá standardu RS232 pro nízkorychlostní NRZ[1], asynchronní komunikace. Protokol RS232 obsahuje mnoho různorodých parametrů. Naši sběrnici ccTalk ve standardním provedení můžeme popsat těmito parametry:

9600 Baudů, 1 start bit, 8 datových bitů, žádný paritní bit a 1 stop bit

[1]Název kódování NRZ pochází z anglického Non Return To Zero. V tomto kódování je jednička „1“ reprezentována konkrétní význačnou hodnotou (například kladným napětím). Nula „0“ je reprezentována jinou význačnou hodnotou (například záporným napětím). Žádné další hodnoty se ve výsledném signálu nevyskytují, neexistuje zde třetí neutrální hodnota. Kvůli absenci neutrální hodnoty nelze toto kódování v základním tvaru použít pro synchronní přenosy, je potřeba přidat synchronizaci například v podobě RLL (run length limited) nebo přídavného signálu hodin.

Signály sběrnice RS232 jako RTS, CTS, DTR, DCD a DSR nejsou ccTalkem podporovány. ccTalk je vyvíjen jako velmi jednoduchý protokol. Formát zpráv je předem stanoven a tudíž nehrozí možnost přetečení dat. Jak vyplývá z výše uvedených parametrů, pro přenos jednoho bajtu informací je nutno fyzicky přenést deset bitů.

Rychlost přenosu byla stanovena kompromisem mezi rychlostí a cenou na 9600 Baudů. Vyšší rychlosti vyžadují výkonnější procesory periferií a s tím jsou spojené větší náklady. Pro kontrolní protokoly sítí v Coin Industry je rychlost 9600 Baudů zároveň nejběžnější rychlostí. Sběrnice podporuje i rychlost 4800 Baudů, ale periferie pracující na této rychlosti se vyskytují jen velmi zřídka.

U zařízení pracujících s „ccTalk over USB“ virtuálními sériovými porty se rychlost pohybuje i na 1 Mb/s a více. Dá se říci, že protokol ccTalk je nezávislý na rychlosti a propustnosti fyzického rozhraní. To umožňuje vývoj cenově efektivních zařízení.

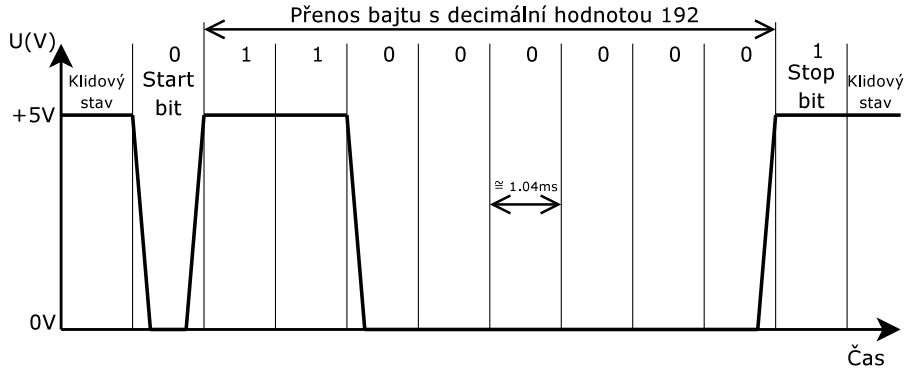
2.4.3 Fyzická vrstva

ccTalk využívá modifikace napěťových vrstev sběrnice RS232. V klidovém stavu je na sběrnici 5 V. Aktivní sběrnice má pak hodnotu 0 V. Povolené rozmezí napěťových úrovní je specifikováno v tabulce 2.2.

Klidový stav	+5 V	Rozsah 3.5 V−5.0 V
Aktivní stav	0 V	Rozsah 0 V−1.0 V

Tabulka 2.2: Napěťové úrovně na sběrnici ccTalk.

Na obrázku 2.1 je uveden příklad přenosu hlavičky příkazu Request Build Code s decimální hodnotou 192.



Obrázek 2.1: Graf přenosu dat v čase.

2.4.4 Struktura zprávy

Protokol podporuje dva druhy kontrolních součtů a rovněž šifrování zpráv. Jedná se o kontrolu jednoduchým součtem a pomocí CRC součtu.

Veškerá komunikace mezi řídícím prvkem a periferií se skládá ze dvou strukturovaných zpráv. První vysílá na sběrnici master. Zařízení, kterému je zpráva určena, vzápětí posílá masteru odpověď. Zprávy mohou, ale nemusí, nést data.

Na rozdíl od jiných protokolů, potvrzovací zpráva dodržuje stejnou strukturu jako ostatní zprávy. Potvrzení jedním bajtem, jak jej implementují jiné protokoly, není považováno za bezpečné. Ani směr zprávy neovlivňuje styl struktury.

Tabulka 2.3 ukazuje strukturu zprávy, která používá jednoduchý kontrolní součet. Tato ukázka obsahuje několik datových bajtů. Pokud zpráva neobsahuje žádné datové bajty, můžeme si je zcela odmyslet a za hlavičku připojit rovnou kontrolní bajt.

Adresa příjemce
Počet datových bajtů
Adresa odesílatele
Hlavička
První datový bajt
...
Poslední datový bajt
Kontrolní bajt

Tabulka 2.3: Struktura zprávy s jednoduchým kontrolním součtem.

Kontrola jednoduchým součtem

Jedná se o typický nulový součet, osmi bitový doplněk všech bajtů ve zprávě, tedy součet všech bajtů ve zprávě musí mít modulo 256 roven 0. Pokud zařízení obdrží zprávu, která nemá součet bajtů dělitelný 256 beze zbytku, je tato zpráva zahozena.

Příklad výpočtu: zpráva [1][0][2][0] bude mít dopočítaný kontrolní součet 253, protože $1 + 0 + 2 + 0 + 253 = 256$

V tabulce 2.4 je uveden příklad potvrzovací zprávy s kontrolou jednoduchým součtem. První bajt obsahuje informaci o adrese příjemce. Druhý přenáší počet datových bajtů. Protože potvrzovací zpráva žádná data nepřenáší, je vyplněn hodnotou 0. Pole „Hlavička“ je nastaveno na hodnotu 0, která signalizuje, že jde právě o typ potvrzovací zprávy.

Adresa příjemce	[1]
Počet dat	[0]
Adresa odesílatele	[2]
Hlavička	[0]
Kontrolní bajt	[253]

Tabulka 2.4: Potvrzovací zpráva s jednoduchým součtem.

Kontrola CRC součtem

ccTalk využívá $16-$bitový CRC-CCITT kontrolní součet, který používá polynomický generátor daný funkcí $x^{16} + x^{12} + x^5 + 1$ a inicializační crc registr je nastaven na hodnotu $0x0000$.

Jelikož velikost CRC součtu je 16 bitů, je potřeba jej rozdělit do dvou bajtů. Bity $7 - 0$ jsou označovány jako LSB. Bity $15 - 8$ jsou pak označovány jako MSB.

Obrázek 2.5 znázorňuje předepsanou strukturu zprávy. Jednou z hlavních změn oproti jednoduchému kontrolnímu součtu je rozdílné využití místa vyhrazeného pro adresu odesílatele, ve kterém se nachází spodní bajt CRC součtu. Pokud je použit tento typ kontrolního součtu, všechna periferní zařízení odpovídají automaticky na adresu 1.

Adresa příjemce
Počet datových bajtů
CRC LSB
Hlavička
První datový bajt
...
Poslední datový bajt
CRC MSB

Tabulka 2.5: Struktura zprávy s CRC součtem.

Adresa příjemce

Rozsah adres je omezen kapacitou jednoho bajtu, který je pro její určení vyhrazen. Rozmezí $0 - 255$ z toho 254 adres pro slave zařízení.

0 :	Broadcast zpráva. Viz. 2.4.4
1 :	standardní adresa řídícího prvku
$2 - 255$:	adresy pro slave zařízení

Zprávy typu Broadcast

Broadcastová zpráva neboli oběžník je taková zpráva, která je určena pro všechna zařízení připojená na sběrnici. Všechna zařízení musí přijímat zprávy určené kromě své vlastní adresy i oběžníkovou adresu 0.

Broadcastové zprávy nalézají uplatnění v aplikacích s více periferiemi. Problém vyvstává s odpověďmi zařízení na sběrnici. Ty mohou vysílat ve stejném čase a tím způsobí kolizi na sběrnici. Odpovědi jsou tedy nečitelné. Aby se master zařízení nemuselo obtěžovat s přijímáním odpovědí, je vhodné jej od sběrnice na určitý čas odpojit nebo přenos na sběrnici ignorovat. Broadcastové zprávy nalézají opravdové využití v několika MDCES příkazech. MDCES příkazy budou vysvětleny v sekci 2.4.6.

Počet datových bajtů

Rozsah je v rozmezí 0 až 252. Tento parametr neoznačuje délku celé zprávy, ale pouze počet datových bajtů zprávou přenášených. Hodnota 0 naznačuje, že zpráva žádná data neobsahuje. Celková délka zprávy bez dat je pak pět bajtů, což je také minimální možná délka. Maximální hodnota 252 znamená, že můžeme očekávat dalších 255 bajtů. K počtu datových bajtů musíme ještě připočíst adresu odesílatele, hlavičku a kontrolní součet. V případě CRC kódování nabývají položky adresa odesílatele a kontrolní součet jiný význam.

Některé aplikace vyžadují přenosy dat ve větším rozsahu než 252 bajtů. V takových případech se data přesouvají po blocích využívajíce sekvence příkazů.

Adresa odesílatele

Rozsah je mezi 1 a 255. Standardní adresa odesílatele jako master zařízení je 1. Když periferie odpovídá, používá svou adresu jako adresu odesílatele. Následující popis se týká rozdílného využití dle typu kontrolního součtu.

Jednoduchý kontrolní součet: Zpráva od řídícího prvku k periferii je nastavena „Cílová adresa = 2, Zdrojová adresa = 1“, bude mít odpověď od periferie ve tvaru: „Cílová adresa = 1, Zdrojová adresa = 2“.

CRC kontrolní součet: Při použití CRC kontrolního součtu nebude použita žádná zdrojová adresa. Obdobný příklad bude vypadat následovně. Zpráva od řídícího prvku k periferii je nastavena „Cílova adresa = 2“ bude mít odpověď od periferie ve tvaru: „Cílová adresa = 1“.

Hlavička

Hlavička může nabývat rozsahu 0 až 255. Hodnota hlavičky rozhoduje o významu zprávy. Money Controls postupně zatřizuje nové hlavičky dle potřeb „Coin Industry“ do nových verzí protokolu. V současné době je velká podpora zařízení typu mincovka, akceptor a hopper.

Hlavička, která nese hodnotu 0, označuje zprávu typu „odpověď“. Takovou zprávu může zaslat pouze periferie řídícímu prvku.

Data

Data nemají žádná striktní omezení. Mohou nabývat hodnoty 0 až 255. Mohou přenášet např. binární, ASCII nebo BCD data. Záleží pouze na typu příkazu specifikovaného hlavičkou, jaký význam se datům přiřadí a jak se použijí.

2.4.5 Příklad komunikace

V této části je podrobně rozebrán dotaz na výrobce periferie. Master se dotazuje mincovky na adrese dvě příkazem „Request Manufacturer Id“, kterému odpovídá hlavička 246. Komunikace používá jednoduchý kontrolní součet.

Dotaz:

Adresa příjemce	[2]
Počet dat	[0]
Adresa odesílatele	[1]
Hlavička	[246]
Kontrolní bajt	[7]

Tabulka 2.6: Příkaz Request Manufacturer Id.

Odpověď je jasně označená hlavičkou 0. Adresu příjemce je nastavena na hodnotu 1, což je původce dotazu. Zpráva přenáší 14 datových bajtů obsahující ASCII řetězec „Money Controls“.

Odpověď:

Adresa příjemce	[1]
Počet dat	[14]
Adresa odesílatele	[2]
Hlavička	[0]
Data	[77] [111] [110] [101] [121] [32] [67] [111] [110] [116] [114] [111] [108] [115]
Kontrolní bajt	[115]

Tabulka 2.7: Odpověď Request Manufacturer Id.

2.4.6 Multi-Drop

ccTalk rozhraní bylo navrhnut tak, aby mělo možnost připojit více zařízení s minimálním úsilím. V topologii připojení není žádné omezení. Jednotlivá zařízení mohou být topologicky připojena např. v jedné řadě, kruhu, hvězdě nebo stromu. Taková rozhraní jsou obvykle pojmenovaná jako Multi-Drop.

Multi-Drop rozhraní je složitější než jednoduché propojení jednoho master a jednoho slave zařízení. V sítích s více slave zařízeními je hlavní problém adresování. Každé slave zařízení musí mít svoji unikátní adresu na sběrnici, aby celý systém fungoval správně.

Pokud se na jedné sběrnici vyskytuje pouze jedno zařízení určitého typu, problém nenastává, protože každý typ zařízení implicitně používá jinou adresu. Mincovka používá standardně adresu 2, hopper adresu 3, akceptor adresu 40, atd. V případě kdy máme na sběrnici více zařízení jednoho typu, musíme zajistit, aby každá periferie měla unikátní adresu. Takové případy nejsou obvyklé, ale ve speciálních situacích mohou mít smysl. Některá zařízení podporují nastavení adresy pomocí DIP switche nebo pomocí využití jiného pinu v konektoru.

Pokud nelze využít některou z výše uvedených metod, ccTalk nabízí dynamické adresování. To znamená, že adresa zařízení může být softwarově změněna. Mnoho z Multi-Drop sítí je deterministických, ale k poskytování plně Multi-Drop sítí slouží MDCES příkazy.

MDCES příkazy

Zkratka vychází z anglického *Multi-Drop Command Extension Set*, což znamená sada rozšiřujících příkazů Multi-Drop. Tyto příkazy přidávají funkce potřebné v Multi-Drop aplikacích. Některé z příkazů ale z principu své funkčnosti neodpovídají standardnímu formátu ccTalk zpráv.

Příkaz 253**, Address Poll:** Tento příkaz je určen k rozpoznávání zařízení na sběrnici. Každé zařízení reaguje vrácením své adresy. Aby se zabránilo kolizi je zpátky vrácen pouze adresní bajt. Odpověď na tento příkaz neodpovídá standardnímu formátu ccTalk. Doba trvání příkazu se na master zařízeních nastavuje na 1.5 s. Do vypršení

tohoto limitu by se na sběrnici měly objevit všechny adresy, na kterých existují zařízení.

Slave odpovědní algoritmus: proměnná *addr* obsahuje adresu periferního zařízení.

1. Periferie vypne přijímací port
2. Počká $4 * addr$ ms
3. Vyšle bajt obsahující svou adresu [*addr*]
4. Počká $1200 - (4 * addr)$ ms
5. Zapne přijímací port

Příkaz 252**, Address Clash:** Tento příkaz je určen k rozpoznávání zda více zařízení sdílí jednu adresu. Příkaz se posílá přímo na určenou adresu. Opět jak tomu bylo v případě Address Poll odpověď na tento příkaz neodpovídá standardnímu formátu ccTalk zpráv.

Slave odpovědní algoritmus: proměnná $r = rand(256)$ obsahuje adresu periferního zařízení. Odpověď je velmi obdobná jako u příkazu Address Poll s tím rozdílem, že periferie místo čekání $4 * addr$ ms generuje náhodné číslo v rozsahu $0 - 255$ a tím násobí konstantu 4 místo adresy. Tím s vysokou pravděpodobností dosáhne odeslání adresy v jinou dobu než ostatní zařízení se stejnou adresou. Pro zmenšení pravděpodobnosti kolize lze příkaz opakovat.

1. Periferie vypne přijímací port
2. Počká $4 * r$ ms
3. Vyšle bajt obsahující svou adresu [*addr*]
4. Počká $1200 - (4 * r)$ ms
5. Zapne přijímací port

Příkaz 251**, Address Change:** Jak název napovídá, příkaz je určen ke změně adresy. Posílá se zařízení s příslušnou adresou a jako jeden datový bajt se uvádí nová adresa. Zařízení odpoví potvrzovací zprávou a změní svou adresu.

Příkaz 250**, Address Random:** Tento příkaz se opět posílá zařízení na určité adrese. Příkaz nenese žádná data. Zařízení odpoví potvrzovací zprávou a náhodně si zvolí novou adresu. Periferie nesmí mít adresu 0, která je určena pro Broadcast a adresu 1 určenou pro master zařízení.

Typická zařízení

Sběrnice ccTalk je dnes rozšířená jako jeden ze standardů v „Coin Industry“. Typickými periferiemi, používající tento protokol jsou například Mincovka (anglicky Coin Acceptor), která slouží k přijímání hotovosti ve formě mincí. Akceptor (anglicky Bill Validator) se využívá hlavně pro příjem bankovek a zařízení zvané Hopper se stará o vyplácení mincí.

2.5 Programování v systémech Linux

Tato část zprávy se stručně zmiňuje o možnostech programování v prostředí systémů Unix. Jedná se hlavně o přístup ke speciálním zařízením jako k obyčejným souborům podporujícím zápis a čtení.

Jednotné prostředí

Základem jednotného prostředí systémů Linux je myšlenka existence jednotného programovacího jazyka, kterým je jazyk C. Programy tvoříme v jazyce C, přičemž využíváme služeb jádra operačního systému. Na tomto principu dosáhneme jednoduché přenositelnosti programu.

V systémech Linux jádro skrývá rozdíl mezi používáním souborů a vnějších zařízení. Některé soubory lze pouze číst, např. klávesnice, do některých lze pouze zapisovat, např. tiskárna, do dalších lze zapisovat i z nich číst. Vnější zařízení jsou chápana jako speciální soubory, které jsou většinou umístěny v adresáři /dev. Každý takový soubor má majoritní a minoritní čísla, které odkazují na příslušný řadič v jádře.

Programování sériového rozhraní

Jako ke každému souboru se dá k rozhraní přistupovat pomocí otevřeného file deskriptoru. Funkce `open` je definována takto: `int open(const char *pathname, int flags)`. Parametrem `pathname` předáme funkci jméno souboru a ona jej otevře a nastaví dle parametru `flags`. Pokud funkce proběhne úspěšně, vrací otevřený file deskriptor. Tomu můžeme nastavovat další libovolné vlastnosti pomocí funkce `fcntl()`.

Režimy čtení

Rozlišujeme dva základní režimy čtení. Blokující režim funguje na principu čekání na data. Pokud jsou v otevřeném file deskriptoru data, funkce `read()` je načte do maximálního počtu, předaného parametrem funkce. Pokud však ve file deskriptoru žádná data nejsou, funkce bude čekat až se nějaká data objeví. Jedná-li se o file deskriptor ukazující na soubor, který je již celý přečtený, systémové jádro pošle speciální hodnotu `EOF`, čili konec souboru. Ovšem pokud file deskriptor ukazuje například na sériový port, funkce žádnou speciální hodnotu neobdrží, protože systém nemůže vědět jestli se na sběrnici nebudou nacházet další data. Délku čekání lze za určitých podmínek nastavit standardní funkcí `tcsetattr()`. Minimální jednotkou času je u čekání desetina sekundy.

U neblokujícího režimu funkce `read()` požádá systémové jádro o data v přijímacím zásobníku. Pokud žádná data nejsou k dispozici, funkce vrací hodnotu -1 a nastavuje *errno* na hodnotu *EAGAIN*. Dále záleží na programátorovi, jak s tímto stavem naloží.

Návrhové vzory

Návrhové vzory – anglicky Design Patterns – se vztahují k obecnému objektově orientovanému programování. Představují řešení problému, které se využívá při návrhu programů. Návrhový vzor není knihovnou nebo částí kódu jenž by se dal vložit do programu, ale spíš popis řešení nebo šablona, která může být použita v různých situacích.

Základními typy vzorů jsou například „vytvářecí“, které řeší problémy související s vytvářením objektů v systému. Další skupinou jsou „strukturální“ vzory zaměřující se na uspořádání jednotlivých tříd.

Návrhový vzor Singleton

Jedná se o vzor, který zajišťuje tvorbu objektu. Smyslem Singletonu je zajistit existenci pouze jedné instance dané třídy a poskytnout k této třídě globální přístup. Je také zapotřebí

zajistit, aby všechna data třídy byla platná po celou dobu existence instance třídy. Tento vzor použiji ve své implementaci.

Singleton není tak jednoduchý návrhový vzor, jak by se mohlo zdát. Vzor musí řešit problémy svázané s destrukcí, vícevláknovostí a další. Existuje více možností řešení jednotlivých problémů. Například „destrukce ostrich“ říká, že pokud vytvoříme statický objekt, můžeme problém destrukce ignorovat. Statická paměť se uvolní automaticky při ukončení procesu. Jiný přístup využívá funkci `atexit()`, u které zaregistruje destrukci instance.

Překlad zdrojových kódů

Aby se dostaly zdrojové kódy do podoby spustitelných binárních souborů, musí projít značně složitým procesem. Já budu zdrojové kódy kompilovat pomocí GNU Compiler Collection tedy GCC. Zadání hovoří o vytvoření sdílené knihovny hlavních funkcí programu. Taková knihovna pak může být k programu přilinkována staticky při překladu, nebo dynamicky vždy po spuštění.

Kompiler GCC umožňuje vytvoření dynamické knihovny. Jednotlivé moduly musíme přeložit s parametrem −fPIC a celkovou knihovnu pak vytvoříme parametrem −shared.

Kapitola 3

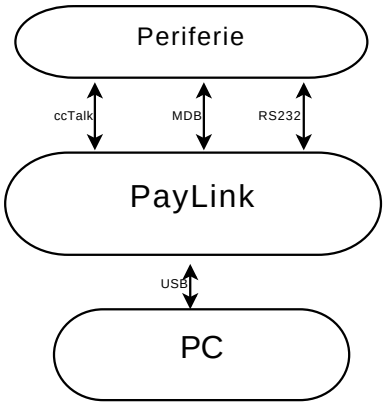
Návrh řešení

Cílem této kapitoly je přiblížit čtenáři zvolený postup a další možné varianty jak lze téma práce vyřešit. První problémem k řešení byl návrh spojení sběrnice s řídící jednotkou. V mém případě s PC. Protože standardní PC nedisponují komunikačním rozhraním pro sběrnici ccTalk musel jsem navrhnout alternativy. Jednou z nich bylo použití komunikačního mezičlánku PayLink, který se dá propojit s PC pomocí sběrnice USB. Další možnost byla připojení sběrnice ccTalk převodníkem přímo k sériovému portu počítače.

3.1 Systém PayLink

PayLink je kompaktní systém vyvíjený firmou Money Controls, který nabízí spojení mezi PC a zařízeními používanými v „Coin Industry“. Je stavěn na velké množství periferií a má možnost periferie připojovat různými druhy sběrnic. Kromě podpory sběrnice ccTalk umožňuje komunikovat např. protokoly ID003, MDB, Ardac 2 či RS232.

Jeho levnější varianta *PayLink Lite* má možnost připojení k periferiím pouze přes sběrnici ccTalk. Oproti své propracovanější verzi nepodporuje tak širokou škálu zařízení, hlavně hopperů. Na obrázku 3.1 je znázorněno blokové použití prvku. V případě verze Lite je pro komunikaci s periferiemi k dispozici pouze rozhraní ccTalk.



Obrázek 3.1: Blokové schéma použití prvku PayLink.

Důvodů, proč jsem některý z těchto prostředků nevyužil ke své práci je několik. Jednak je nevýhodou samotné používání další „krabičky“, která může působit komplikace. Cena

zařízení, větší počet kabeláže a konektorů jsou dalšími faktory hrajícími proti. Pro ovládání systému PayLink jsou napsány knihovní funkce v jazyce C. Tyto kódy vyvíjí společnost Money Controls a vydává je v binární podobě.

3.2 Protokol RS232

Název se vyvinul z anglických slov *Recommended Standard*. Poslední platný standard vyšel v roce 1969 a nese označení RS-232C. Protokol definuje jak přenášet sériově binární data mezi zařízeními DTE a DCE[1].

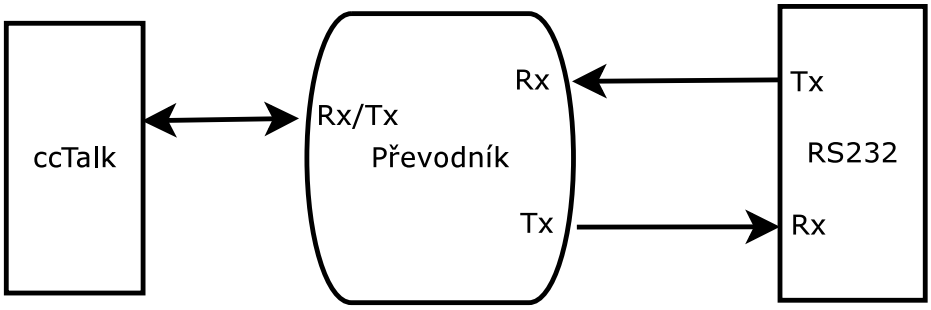
V novodobých počítačích se často sériové rozhraní nahrazuje modernějším USB[2]. Univerzální sériové rozhraní v porovnání se standardem RS-232 je rychlejší, využívá nižší napěťové úrovně a je uživatelsky jednoduší.

Protokol podporuje jak synchronní, tak asynchronní přenos. Data jsou reprezentována jako časová posloupnost bitů. Standard definuje řadu kontrolních mechanismů, které jsou používány ke spojení DTE a DCE. Každý datový nebo kontrolní obvod pracuje v jednom směru. Protože jsou obvody pro vysílání a příjímání dat odděleny, rozhraní může být využíváno v režimu Full Duplex[3].

Pořadí vysílání datových bitů postupuje od nejméně významného bitu LSB po nejvýznamnější bit MSB. Počet datových bitů není striktně stanoven. Obvykle se volí možnost osmibitových celků.

3.3 Převodník ccTalk na RS232

Převodník je velmi jednoduché elektronické zařízení, založené na integrovaném obvodu řady MAX232. Kvůli podobnosti těchto protokolů převodníku stačí upravit velikosti napěťových úrovní. Oproti předešlé variantě PayLinku, kdy počítač musí využít předepsaných funkcí ke komunikaci, je zde sběrnice přímo dostupná na sériovém portu počítače.



Obrázek 3.2: Blokové schéma použití převodníku ccTalk na RS232.

[1] Označení DTE a DCE zavedla firma IBM. DTE je zkratka pro Data Terminal Equipment v telekomunikačních technologiích koncové zařízení. DCE je zkratka pro Data Communications Equipment a označuje zařízení, které řídí zejména hodinový signál pro zařízení DTE.

[2] USB - Universal Serial Bus) je univerzální sériová sběrnice. Nahrazuje dříve používané způsoby připojení pro běžné druhy periferií.

[3] Full Duplex označuje způsob komunikace, kdy mohou být data přenášena oběma směry současně.

Jednou z vlastností sběrnice ccTalk je obousměrný datový vodič. Převodník se s touto specialitou vypořádává propojením vodičů Rx a Tx[4] na straně sběrnice ccTalk. Všechny signály, které převodník vyšle na sběrnici ccTalk okamžitě přijímá a překládá zpět na RS232. Tímto způsobem počítač obdrží kopie odeslaných signálů. Tato vlastnost se může využít pro testování připojení převodníku. Software ovšem nesmí tyto signály zpracovávat jako odpovědní zprávu.

Díky převodníku na sběrnici RS232 se naskýtá další jednoduchý převod na rozhraní USB. Levným převodníkem jsme schopni provozovat ccTalk i na počítačích, které z různých důvodu nemají k dispozici standardní sériový port. Pro převodník na protokol USB lze v Linuxových systémech využít jaderný modul *ftdi_sio*, který jej automaticky identifikuje a vytvoří pro něj speciální sériové rozhraní pojmenované /dev/ttyUSBx, kde x je pořadové číslo převodníku.

3.4 ccTalk API

API je zkratka pro anglická slova *Application Programming Interface*, což znamená sbírku funkcí, procedur či tříd řešících určitou problematiku. Ty mohou programátoři využívat jako hotová řešení.

V době zpracování této práce neexistovalo žádné volně dostupné programátorské prostředí, které by zajišťovalo komunikaci přímo se sběrnicí ccTalk, proto jsem se jej rozhodl vytvořit.

Základem mnou vytvořeného rozhraní je jádro[5], které bude zajišťovat odesílání a přijímání dat ze sběrnice. Na takové jádro by pak měly navazovat další funkce, které budou jeho služby zprostředkovávat okolí s obecným formátem vstupů a výstupů.

Softwarové jádro aplikace by mělo splňovat určité podmínky práce se sběrnicí ccTalk. Když navrhujeme rozhraní programu, musíme přemýšlet o jeho použití. Například u obsluhy více zařízení na sběrnici je dost možné, že programátor bude chtít naprogramovat aplikaci paralelně. Z toho vyplývá, že implementace by měla zabránit možnosti přístupu ke sběrnici více procesům nebo vláknům současně.

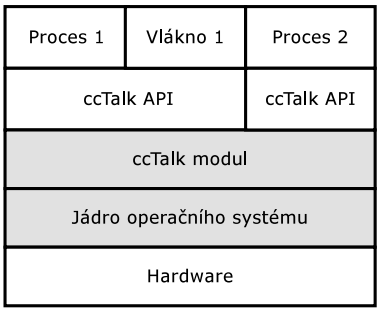
Variant, které se nabízí na návrh softwarové jádra rozhraní, je několik. Má-li být varianta přípustná, musí splňovat pravidla potřebná k bezchybné práci se sběrnicí.

Jaderný modul

Jednou z možností je implementace podpory protokolu ccTalk přímo do jádra operačního systému. Musíme vzít do úvahy, že sběrnice může být připojena k počítači na libovolném sériovém nebo USB portu. Jaderný driver ftdi_sio z každého připojeného USB převodníku vytvoří speciální zařízení. Jádro ovšem nemá žádnou možnost zjistit, jestli je na sériovém portu připojena sběrnice ccTalk, jelikož žádné periferní zařízení na sběrnici nemůže samo zahájit komunikaci. Jádru by se tedy musel předávat parametr, na kterém sériovém rozhraní se sběrnice opravdu nachází. Takovou situaci lze v reálných aplikacích vyřešit jednodušeji na vyšších úrovních. Diagram znázorňující posloupnost propojení vrstev aplikace a fyzického hardwaru je ukázán na následujícím obrázku 3.3. Součásti jádra operačního systému jsou podbarveny.

[4] Rx, Tx jsou vodiče na kterých se vysílají Tx (Transmit) a přijímají Rx (Receive) data. Tyto vodiče má převodník pro každou sběrnici odděleně.

[5] Jádro rozhraní není zaměnitelné s jádrem operačního systému.

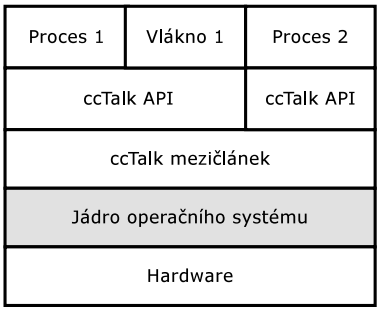


Obrázek 3.3: Diagram přístupových vrstev varianty „Jaderný modul“.

Síťový démon jako jádro rozhraní

Další z nápadů bylo vytvořit komunikační mezičlánek, který by zprostředkovával sběrnici ccTalk ostatním procesům. Démon by mohl pracovat jako TCP či UDP server. Iterativní[6] verze serveru by jednoduše řešila problém paralelního přístupu ke sběrnici.

Ostatní funkce rozhraní zprostředkovávající služby jádra by si musely pro každou operaci se sběrnicí vytvářet nové síťové spojení. Také by bylo zapotřebí navrhnout komunikační protokol používaný mezi démonem a funkcemi rozhraní. Na obrázku 3.4 je blokové schéma propojení vrstev softwarové aplikace s hardwarem. Na rozdíl od řešení jaderným modulem může toto řešení vzniknout v čistě uživatelském prostředí.

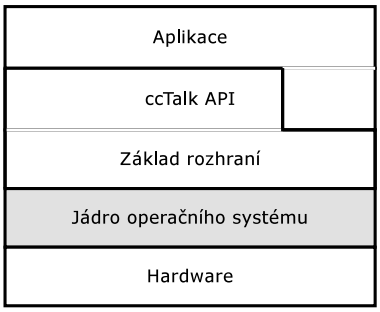


Obrázek 3.4: Diagram přístupových vrstev varianty „Síťový démon“.

Jádro součástí API

Varianta implementování jádra přímo do knihoven rozhraní poskytuje řešení bez nutnosti spouštění externích programů či zásahů do jádra operačního systému. Aplikace využívající takové rozhraní pak může přistupovat k povoleným službám jádra rozhraní přímo. Diagram 3.5 ukazuje možnosti využití a návazností od softwarových až po hardwarovou vrstvu.

[6] Iterativní server je takový, který příchozí požadavky vyřizuje postupně. Další verze je konkurentní, která příchozí spojení obsluhuje paralelně.



Obrázek 3.5: Diagram přístupových vrstev varianty „Jádro součástí API“.

3.5 Paralelní přístup ke sběrnici

U implementace „jádro součásti API“ a varianty „mezičlánkového řešení“ je potřeba vyřešit paralelní přístup ke sběrnici. Sběrnice v tomto případě představuje sdílený zdroj.

Představme si situaci, kdy bude spuštěno více instancí démona a tito démoni budou připojení na stejné sériové rozhraní. Stejný stav může nastat u několika běžících aplikací používajících opět stejné sériové rozhraní. V kritické době, kdy jedna aplikace využívá sběrnici (odesílá zprávu, čeká na odpověď nebo přijímá odpověď), může druhá aplikace začít provádět své operace. Po takovém střetu budou dotazy obou aplikací neúspěšné.

Ve většině případů mají aplikace vysoké požadavky na dostupnost sběrnice. Stane-li se, že musí čekat na přidělení sběrnice delší čas, je pro ně výsledek bezcenný. Programátor by proto měl mít pod kontrolou veškerou práci se sběrnicí. Bude tedy zapotřebí, implementovat přístup k sériovému rozhraní pouze pro jeden proces.

Rozdíl mezi přístupy výše zmíněných variant je ve způsobu používání jádra. U „síťového mezičlánku“ bude jeho jediná instance mít zaručen exkluzivní přístup k sériovému portu. U implementace „jádro součástí rozhraní“ bude kromě exkluzivního přístupu ještě potřeba ošetřit paralelní přístup k samotnému jádru rozhraní.

3.6 Objektově orientovaný návrh

Tato sekce se věnuje objektově orientovanému návrhu řešení varianty „jádro součástí API“. Jsou zde popsány finální třídy. Není žádoucí v této práci popisovat všechny varianty tříd či původně vyvíjené modulární řešení.

V zadání této práce jsou specifikovány možné implementační jazyky na jazyk C a C++ [4] [5]. Na začátku implementace softwaru jsem chtěl používat výhradně jazyk C. Ve vývojových cyklech jsem postupně přešel na jazyk C++. Vedla mě k tomu složitost a nepřehlednost modulárního způsobu zpracování. Mnohem elegantnější bylo použít objektově orientovaný návrh.

Jazyk C++ poskytuje řadu výhod, které lze při návrhu a programování využít. Jako příklad uvedu jmenné prostory. Veškeré definice tříd a datových typů, potřebných pro činnost rozhraní, jsou vnořeny do speciálního jmenného prostoru.

Základní třídou, která se věnuje přímé práci se sériovým rozhraním, je třída `bus`. Její UML diagram je na obrázku 3.7. Další spolupracující třídou je `message` - viz. obrázek 3.8. Ta představuje jednotlivé zprávy zasílané a přijímané sběrnicí. Instance třídy `message` používají třídu `data` - obrázek 3.9. `Data` obsahují datové bajty přenášené jednotlivými zprá-

vami. Vše zastřešuje třída `device` - obrázek 3.6, kterou si můžeme představit jako výchozí třídu pro zařízení připojené na sběrnici. Programátor využívající rozhraní si může třídu `device` zdědit a implementovat dle svých potřeb další metody. Pro ulehčení programátorovy práce je připraven modul `commands`, obsahující sadu funkcí řešící příkazy protokolu ccTalk. Všechny UML diagramy tříd jsou umístěny na konci kapitoly.

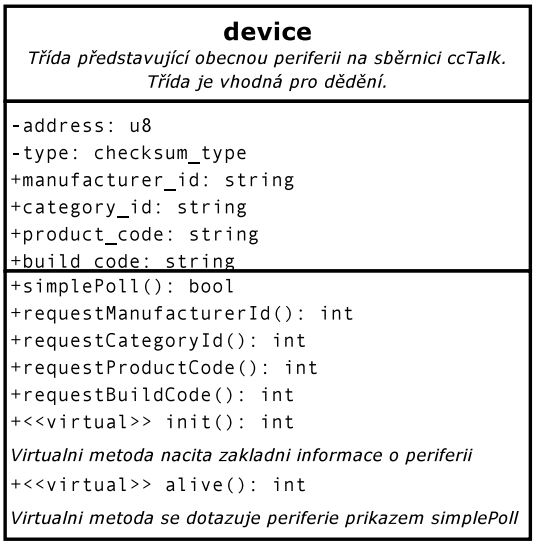
3.7 Zpracování příkazů protokolu ccTalk

Příkaz se z pohledu zařízení master skládá ze dvou zpráv. První putuje sběrnicí směrem ke slave zařízení.

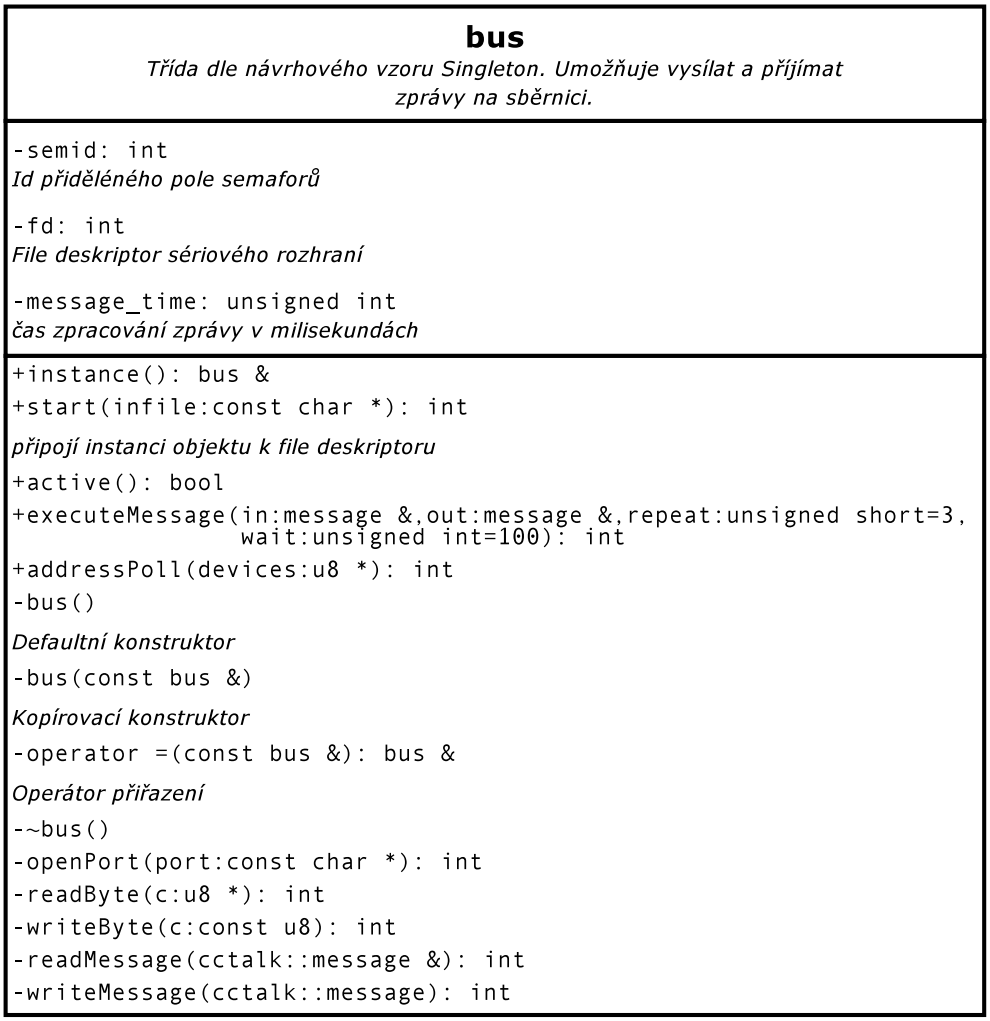
Čas čekání na odpověď periferie se může značně lišit dle typu dotazu. Software zařízení by měl být napsán tak, aby master musel čekat jen nezbytně nutnou dobu. Nejjednodušší příkaz je *SimplePoll*, který se používá pouze na zjištění zda je zařízení zapnuto a pracuje. Odpověď na tento příkaz by mohla být kratší než 10 ms. Řídící program by měl zohledňovat délku čekání na odpověď dle daného příkazu.

Aby bylo příjímání zprávy optimalizováno, bude muset probíhat dle určitého algoritmu. Můžeme využít předem známou strukturu zprávy. Prvním bajtem přichází hlavička zprávy. Druhým počet datových bajtů. Ostatní bajty ve zprávě jsou adresa odesílatele či první část kontrolního součtu, hlavička zprávy, případná data a na konci druhá část nebo celý kontrolní součet (záleží na typu součtu). Pokud přijmeme první dva bajty, můžeme z nich dopočítat požadovaný počet následujících bajtů a ty také postupně načíst. Pokud čekáme na další bajt uprostřed zprávy, existuje výrobcem doporučená doba (50 ms), po které můžeme načítání ukončit a zprávu zahodit. Abychom zabránili možnosti přijetí poškozených dat, musíme nad zprávou provést kontrolní součet a ověřit tak integritu dat.

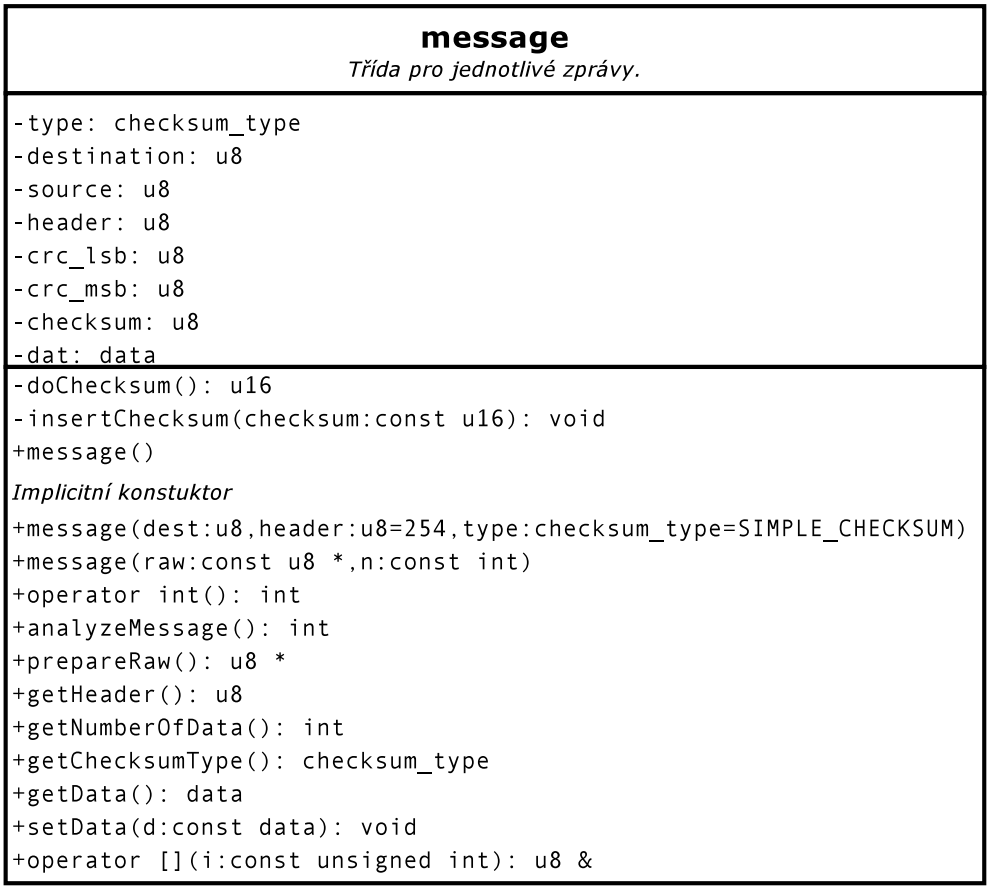
UML diagramy tříd



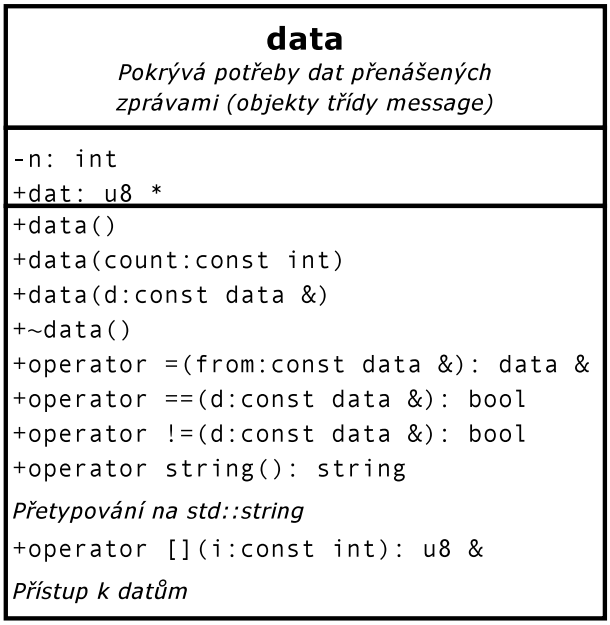
Obrázek 3.6: UML diagram třídy `device`.



Obrázek 3.7: UML diagram třídy `bus`.



Obrázek 3.8: UML diagram třídy `message`.



Obrázek 3.9: UML diagram třídy `data`.

Kapitola 4

Implementace

Tato kapitola slouží jako přehled důležitých informací o problémech, které jsem musel v průběhu psaní rozhraní a testovacích programů řešit. Také ji lze využít jako podrobnější manuál k používání implementované knihovny.

4.1 Nastavení sériového portu

Zjištění správného nastavení sériového portu mi při vývoji zabralo značné množství času, proto jsem se rozhodl jej pečlivěji popsat. Vědomosti jsem čerpal mimo jiné ze zdroje [6].

Základní rozhodnutí o přístupu k otevřenému portu je, zda bude používat blokovací či neblokovací čtecí režim. Vlastnosti obou režimů jsou popsány v sekci 2.5.

Načítáme-li data blokovacím režimem ze sběrnice, na které žádná data nejsou, funkce čeká až do příchodu dalších dat nebo do uplynutí maximální doby, kterou lze nastavit pomocí funkce `tcsetattr()`. Tento přístup by byl použitelný, kdyby se dala maximální doba čekání nastavit na kratší časový úsek než desetinu vteřiny. Proto jsem zvolil neblokovací režim, který mi dovoluje zkrátit nadbytečné časy na minimum.

Operační systém Linux poskytuje mnoho možností nastavení sériových portů. Ty jsou definovány ve struktuře `struct termios`. Struktura se skládá z následujících proměnných. `c_iflag` obsahuje příznaky nastavení vstupu, `c_oflag` obsahuje příznaky nastavení výstupu, `c_cflag` obsahuje příznaky kontroly a další. Jelikož sběrnice ccTalk pracuje v binárním režimu, musíme nastavit port do *non-canonical* modu zpracování. Ten zaručuje, že zprávy nebudou řádkově zpracovávány[1].

Pro změnu nastavení musíme vytvořit proměnou typu `struct termios`. Je dobrým zvykem do ní načíst původní nastavení portu funkcí `tcgetattr()`. K tomu jsou zapotřebí hlavičkové soubory `termios.h` a `unistd.h`. Funkcí `cfmakeraw()` můžeme nastavit port do takzvaného *raw* režimu, kde operační systém přenášená data ponechává bez dalších úprav. Softwarové řízení toku vypneme pomocí nastavení těchto příznaků: `IXON`, `IXOFF` a `IXANY` v proměnné struktury `termios` `c_iflag` na nulu. Funkce `cfsetispeed()` a `cfsetospeed()` slouží k nastavení rychlosti. V našem případě nastavíme rychlost na 9600 Baudů. Další parametry nastavíme podle potřeb sběrnice ccTalk – žádný paritní bit, jeden stop bit a osm datových bitů. Toho dosáhneme vynulováním příznaků `PARENB`, `CSTOPB` a nastavením `CS8`. Podrobné informace o všech příznacích a struktuře termios lze zjistit např. z Linuxových manuálových stránek [7] nebo z internetových stránek [1]. Nyní můžeme nastavení portu

[1] Řádkové zpracování je takové, kde zprávu ukončuje znak CR či LF. Záleží na druhu operačního systému.

uložit funkcí `tcsetattr()`. Aby se nastavení projevilo okamžitě, musíme funkci předat jako parametr akce `TCSANOW`.

4.2 Jmenný prostor `cctalk`

Celé rozhraní se nachází ve jmenném prostoru `cctalk::`. Kromě dále definovaných tříd, obsahuje jmenný prostor také definice datových typů, výčtů a podobně. Protože jsou třídy součástí jmenného prostoru, mohou ve svých metodách používat datové typy jmenného prostoru cctalk.

Velmi používaný datový typ `cctalk::u8` je definován jako `typedef unsigned char u8;` a používá se pro proměnné, které mají vlastnosti jednoho bajtu. Pro potřeby nejen kontrolních součtů, které mohou být 16-bitové je k dispozici datový typ `cctalk::u16` definovaný příkazem `typedef unsigned short u16`. Typ kontroly zprávy je definovaný výčtem `checksum_type`.

4.3 Třída `bus`

V této sekci si popíšeme hlavní metody a princip činnosti. UML popis třídy je k nalezeni na konci minulé kapitoly 3.7.

Tato třída zajišťuje práci přímo se sériovým rozhraním. Je koncipovaná tak, aby pro celý program existovala pouze jedna instance. Tento požadavek mě přiměl k implementaci třídy pomocí návrhového vzoru Singleton. Vlastnosti Singletonu jsou popsány v sekci 2.5. Přístup k instanci třídy je realizován pomocí metody `instance()`, která vrací referenci na vytvořenou instanci.

Aby mohl programátor instanci oznámit systémovou cestu k zařízení představující sériový port s připojenou ccTalk sběrnicí, musí použít metodu `start()`, které jako parametr předá cestu. Metoda je pouze public rozhraní pro privátní metodu `open_port()`. Ta sériové rozhraní otevře, uzamkne a nastaví parametry pro komunikaci s zařízeními na sběrnici. V případě úspěchu vrací hodnotu 0. Jinak metoda vrátí číslo chybového stavu a vypisuje chybu na standardní chybový výstup.

Pokud bude hlavní program obsahovat vlákna pracující se sběrnicí, doporučuji vytvořit instanci třídy a připojit ji ke sběrnici ještě v hlavním programu před spuštěním vláken.

Hlavní používanou metodou třídy bude `executeMessage()`, která zpracovává příkazy protokolu ccTalk a je definována takto:

```
int executeMessage(message & in, message & out, unsigned short repeat = 3,
unsigned int wait = 100)
```

Tato metoda má dva vstupně výstupní parametry a dva nepovinné parametry, které nastavují počet opakování dotazu v případě neúspěšného zpracování a maximální délku čekání na první bajt odpovědi. Chce-li program provést komunikaci po sběrnici, využívá se výhradně `executeMessage()`. Uvnitř metody se volají další privátní metody: `readMessage()`, `writeMessage()`, `readByte()` a `writeByte()`. Jejich funkcionalitu popisují samotné názvy. Je potřeba zmínit, že metody využívají kromě systémových funkcí `read()` a `write()` další funkce jako `tcflush()`. Metoda `executeMessage()` vrací počet milisekund zabraných zpracováním dotazu nebo v případě selhání hodnotu 0.

4.4 Třída `message`

Instanci třídy message si můžeme představit jako jednu zprávu definovanou dle protokolu ccTalk. Taková zpráva může být odeslána na sběrnici či z ní přijata.

Třída má nadefinováno pět různých konstruktorů. Implicitní konstruktor vytváří zprávu pro příkaz *Simple Poll* s hlavičkou 254, zdrojovou adresou 1, cílovou adresou 0, jednoduchým kontrolním součtem a žádnými daty. Další konstruktory se využívají pro vytvoření specifičtějších zpráv. Konstruktoru lze parametry předat cílovou adresu, hlavičku, zdrojovou adresu a typ kontroly zprávy. Od parametru zdrojové adresy včetně mohou být tyto parametry implicitně nastavovány. Pro vytváření zpráv přijímaných ze sběrnice je vytvořen speciální konstruktor, který přijímá ukazatel na pole bajtů typu `u8` a celkový počet bajtů v poli.

Každý objekt typu message obsahuje jednu instanci třídy data. Ta bude popsána v příští sekci 4.5. Využívá se pro plnění zpráv užitečnými daty.

Privátní metody `doChecksum()` a `insertChecksum()` se používají při vytváření, pozměňování a kontrole zpráv. `doChecksum()` předává ve své návratové hodnotě vypočítaný kontrolní součet. O tom, zda se jedná o CRC či jednoduchý součet, rozhoduje privátní atribut `checksum_type type`.

Pro získávání informací o zprávě slouží veřejné metody `getData()`, `getNumberOfData()`, `getChecksumType()` a `getHeader()`. Data zprávy lze nastavit pomocí funkce `setData()`.

Přetížený operátor `operator int()` lze využít ke zjištění integrity zprávy. Pokud kontrolní součet neodpovídá obsahu zprávy, je vrácena záporná hodnota. Jinak se vrací hodnota hlavičkového bajtu. Stejnou akci vykonává také veřejná metoda `analyzeMessage()`.

Dvě další metody jsou určeny především pro užití do výrazů podmínek. Jsou to metody `bool isACK()` a `bool isNAK()` vracejí logickou hodnotu *true* jestliže se jedná o zprávu typu ACK *Acknowledge message* či NAK *Not Acknowledge message*. Třída také disponuje přetíženým operátorem `friend std::ostream & operator << (std::ostream &, message &)`, který umožňuje tisknout instance třídy do standardních streamů.

4.5 Třída `data`

Třída disponuje dvěma atributy. `int n` obsahuje počet datových bajtů. Druhým atributem je ukazatel na samotná data `u8 * dat`. Konstruktoru třídy lze předat počet právě těchto datových bajtů, kterými bude instance disponovat.

Data se ukládají na haldě pomocí operátoru `new [ ]`. Počet dat obsažených v instanci třídy lze zjistit pomocí veřejné metody `count()`. Přístup k jednotlivým bajtům umožňuje přetížení operátoru hranaté závorky `operator [] (unsigned int)`, skrze který se lze dostat ke všem datovým bajtům. Pokud je indexován prvek nad rámec přenášených dat, metoda vyvolá výjimku typu `cctalk::error`. Pří zániku objektu jsou data v destruktoru smazána.

Dále má třída nadefinované operátory rovnosti a nerovnosti, operátor přiřazení, operátor přetypování na string a také přetížený operátor pro tisk do standardních streamů.

4.6 Modul `commands`

Ne vždy je vhodné implementovat celý kód objektově orientovaným způsobem. Tento modul obsahuje řadu funkcí, které používají ostatní objekty, ale samy náleží pouze jmennému pro-

storu `cctalk::commands::`. Jsou zde funkce implementující řadu příkazů protokolu ccTalk. Každý příkaz odpovídá určité hlavičce zprávy. Kompletní přehled příkazů a hlaviček můžete nalézt ve třetím dílu manuálu výrobce na stránkách [2].

V této práci jsem se zaměřil hlavně na příkazy podporující zařízení typu mincovka a hopper, protože jsem je měl k dispozici a mohl jsem příkazy při vývoji testovat.

4.7 Třída `device`

Tato třída je určena jako výchozí třída pro dědění objektů představujících fyzická zařízení na sběrnici.

Třída obsahuje chráněné atributy `u8 address` a `checksum_type type`. Má implementovány příkazy pro získání základních informací o periferii. Každé zařízení podporující sběrnici ccTalk musí znát tyto příkazy známé jako implementační minimum. Jde například o příkazy: *Simple Poll*, *Request Manufacturer Id*, *Request Category Id* a další. Metody odpovídající těmto příkazům jsou implementovány voláním funkcí z modulu `commands`. Výsledky ukládají do veřejných atributů např. pro metodu `requestManufacturerId()` je výstupní atribut `std::string manufacturer_id`.

Virtuální metody `init()` a `alive()` jsou určeny pro vytváření polymorfních rozhraní. To umožňuje vytvářet objekty různých odvozených tříd a pověsit je na ukazatel na bázovou třídu (tedy například na třídu `device`). Virtuální metody volané přes ukazatel na bázovou třídu jsou pak vykonávány na skutečném objektu, na který ukazatel ukazuje. Takového chování není možné dosáhnout u standardních metod, ale jen při použití pozdní vazby.

4.8 Programové zabezpečení sdílení sběrnice ccTalk

Snahou bylo vytvořit ccTalk rozhraní tak, aby nezpůsobovalo problémy v případě, že se programátor rozhodne použít více souběžných procesů či vláken.

Jako hlavní část zabezpečení jsem implementoval kritické metody přístupu ke sběrnici tak, aby jej mohl současně využívat maximálně jeden proces či vlákno. Protože jsem do rozhraní nechtěl připojovat další externí knihovní bloky, které obsahují systémy zámků (mutexů), rozhodl jsem se využít možnosti systémových semaforů [3]. Jedná se o standardní POSIX funkce pro získání semaforů `semget()`, manipulaci se semafory `semctl()` a atomickou funkci pro práci se semafory `semop()`.

Třída `bus` využívá semafor pro zpracování zpráv. Pokud je zpráva vykonávána, prochází stavy: odeslání, čekání na odpověď a příjem odpovědi. Po celou dobu vykonávání je nepřípustný jakýkoli zásah cizího vlákna. Konkrétně se jedná o metody `executeMessage()` a také metodu `addressPoll()`, která je určena k rozpoznávání zařízení na sběrnici. O zablokování a uvolnění těchto funkcí se starají privátní metody `getBus()` a `releaseBus()`. Je-li sběrnice zaneprázdněna, metoda `getBus()` čeká tak dlouho, dokud nedojde k jejímu uvolnění. Zabraný počet milisekund čekáním pak vrací návratovou hodnotou.

V celém rozhraní jsou pro čekání používány funkce `msleep(unsigned long)`, která jako svůj parametr přebírá čas v milisekundách. Funkce `msleep()` je zprostředkovatel systémové funkce `nanosleep()` jež je bezpečná ve vícevláknovém prostředí.

Kapitola 5

Testování

Testování probíhalo v rámci vývoje celé práce. Pro potřeby testování jsem měl zapůjčeny dvě periferie. Standardní mincovku SR3 od společnosti Money Controls a Universal Hopper vyrobený stejnou společností.

Na počátku jsem musel řešit hardwarový problém s převodníkem sběrnice ccTalk a RS232. Jeho elektrotechnický nákres naleznete v příloze B.1. Závady se skrývaly ve špatném překreslení schématu na tištěný spoj. Logickým analyzátorem jsem závady odhalil a drobných úpravách se mi podařilo převodník zprovoznit.

K rozvodu sběrnice k periferiím jsem měl k dispozici ccTalk hub. Tento hub funguje pouhým propojením signálů více konektorů. Obrázek některých komponent a celé testovací sady naleznete v příloze C.

Implementaci jádra rozhraní jsem pečlivě otestoval na uvedeném hardwaru. Většina funkcí implementovaných v modulu `commands` je taktéž vyzkoušena v reálných podmínkách. Příkazy, které mnou testované zařízení nepodporují, vyzkoušené nejsou, ale z důvodu podobnosti s ostatními nepředpokládám problémy při použití.

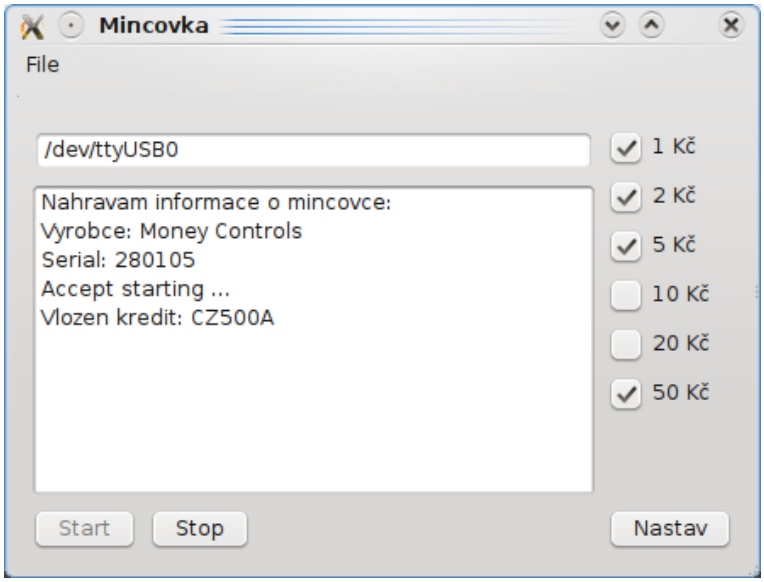
Předmětem testování bylo mimo jiné i přepínání kontextu operačním systémem ve více vláknovém prostředí. První implementace zabezpečení sdíleného zdroje (sběrnice) byla realizována pomocí testování a nastavování atributu třídy `bus` datového typu `bool`. I když se můžeme spolehnout na atomičnost operace zápisu a čtení datového typu bool na většině počítačových architektur, existuje zde možnost přepnutí kontextu mezi otestováním a nastavením proměnné. Tento problém byl vyřešen pomocí implementování POSIX semaforů, kde operační systém garantuje, že funkce `semop()`, která semafor otestuje i nastaví, je atomická. Třída `bus` již první způsob zabezpečení neobsahuje.

Testovací programy

Součástí odevzdávané práce je i CD-ROM, na kterém jsou mimo jiné i zdrojové kódy a Makefily potřebné k přeložení dvou ukázkových programů, na kterých jsem také rozhraní testoval. V rámci vývoje rozhraní jsem udělal více testovacích programů, které sloužily k otestování určitých částí knihovny. Kód těchto programů jsem časem měnil a přizpůsoboval si ho aktuálním potřebám. Tento kód nezahrnuji do odevzdávaných souborů na CD-ROM.

Program Mincovka

Program „Mincovka“ má jednoduché grafické rozhraní, kde se dá nastavovat a zakazovat příjímání mincí určené hodnoty. Obrázek GUI je na obrázku 5.1. Program se spouští bez parametrů. Do editovatelného řádku v horní části programu se zadá cesta k sériovému rozhraní. Tlačítko „Start“ program připojí ke sběrnici. Program očekává na sběrnici připojenou periferii mincovka na logické adrese dvě. Program načte informace o mincovce. Pomocí tlačítka „Nastav“ lze mincovce povolovat a zakazovat přijímání mincí. Aplikace vypisuje informace o nastavení a přijatých mincích do okna pod adresou sériového portu.



Obrázek 5.1: GUI programu Mincovka.

Program InOut

Program „InOut“ představuje obsluhu komponent mincovka a hopper. Aplikace nastaví mincovku pro příjem všech českých mincí. Hopper za každých vložených 10 Kč vyplácí jednu minci. Program se spouští v konsolovém prostředí a přijímá jeden parametr s cestou k sériovému portu. InOut kontroluje množství vložených mincí a v případě překročení částky 10 Kč pouští vlákno pro vyplacení potřebného množství mincí. Program je možno korektně ukončit stiskem libovolné klávesy.

Kapitola 6

Závěr

Hlavním cílem této práce bylo vytvoření programového rozhraní pracujícího se sběrnicí ccTalk pro operační systém Linux. Přínosem je vybudování volně dostupného knihovního rozhraní, které bylo dříve postrádáno. Software jsem založil na objektově orientovaném přístupu. Výhodou tohoto řešení je snadná možnost rozšíření a také jednoduché používání v dalších aplikacích. Systém jsem otestoval na zapůjčených periferiích - mincovka, hopper.

Za další přínos práce můžeme považovat český popis sběrnice ccTalk v úvodu této zprávy. Lze se z něj dozvědět základní informace potřebné pro práci se sběrnicí.

Jako rozšíření tohoto projektu by bylo vhodné doplnit jej o možnost šifrování zpráv, vytvořit monitor vytížení sběrnice ccTalk, zpracovat manuálové stránky rozhraní či rozšířit modul commands o podporu dalších ccTalk zpráv.

Při zpracování tohoto tématu jsem si rozšířil přehled o sériových sběrnicích a jejich programování v prostředí operačního systému Linux. Blíže jsem se seznámil s možnostmi POSIX semaforů a meziprocesové komunikace. Také pro mě bylo důležité „oprášení“ elektrotechnických znalostí, které jsem využil k řešení hardwarových problémů.

Literatura

[1] The Single UNIX Specification, Version 3. `http://www.unix.org/single_unix_specification/`, 2004.

[2] ccTalk Generic Specification. `http://www.cctalk.org/`, 2005.

[3] Brian Hall: Beej's Guide to Unix Interprocess Communication. `http://www.ecst.csuchico.edu/~beej/guide/ipc/`, 2007.

[4] Bruce Eckel, C. A.: *Myslíme v C++*. Grada, 2000, iSBN 80-247-9009-2.

[5] Bruce Eckel, C. A.: *Myslíme v C++, 2. díl*. Grada, 2005, iSBN 80-247-1015-3.

[6] Gary Frerking, Peter Baumann: Serial Programming HOWTO. `http://www.tldp.org/HOWTO/Serial-Programming-HOWTO/`, 2001.

[7] Manuálové stránky: termios. `man 3 termios`.

Příloha A

Obsah CD

Technická zpráva

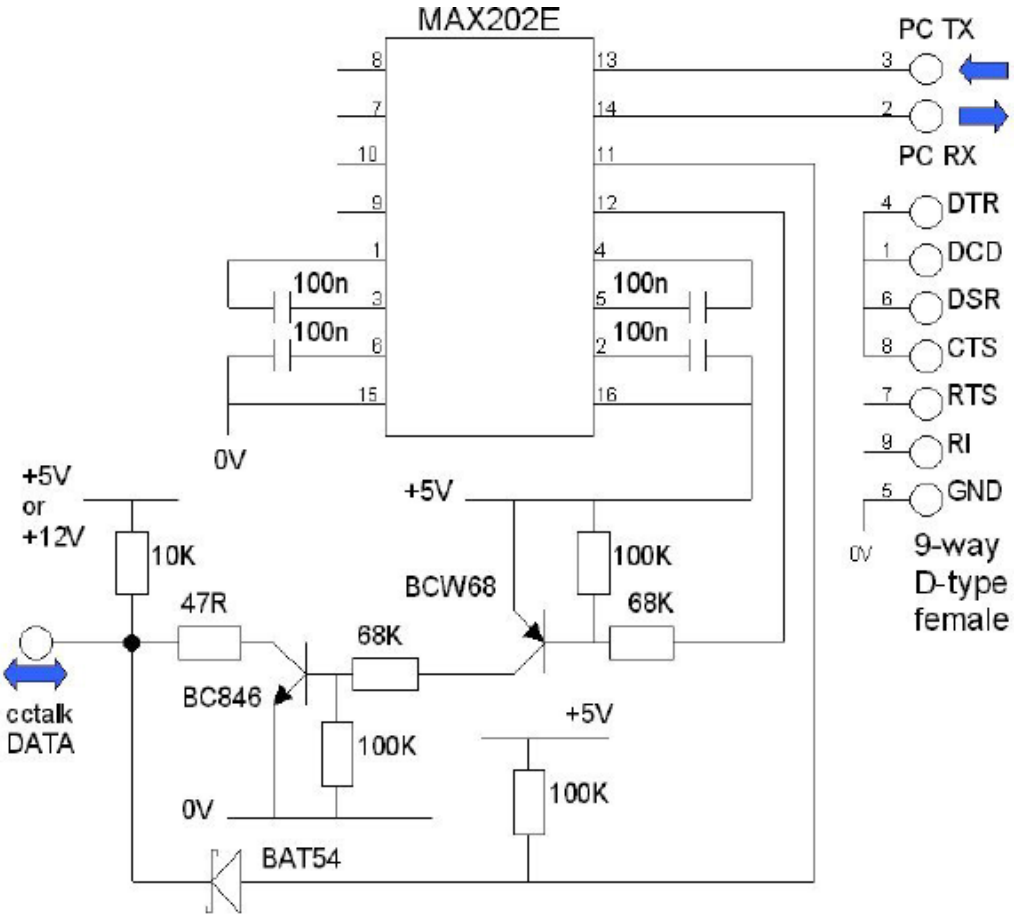
Zpráva ve formátu PDF se nachází v adresáři `zprava/cctalk.pdf`. Technická zpráva byla zpracována dle platné šablony FIT 2009 pro Tex. Zdrojové soubory zprávy ve formátu Tex jsou v adresáři `zprava/tex/`. Soubor s obsahem zprávy se jmenuje `obsah.tex`.

Zdrojové kódy

Zdrojové soubory se nacházejí v adresáři `sources/`. V tomto adresáři se nalézájí zdrojové kódy knihovního rozhraní `LibccTalk` a ukázkové programy `InOut` a `Mincovka`. V každém z těchto adresářů je soubor `README` obsahující návod na kompilaci a základní informace o programech.

Příloha B

Schéma převodníku ccTalk na RS232.



Obrázek B.1: Elektrotechnické schéma převodníku.

Příloha C

Testovací hardware

Obrázek C.1: Univerzální Hopper.

Obrázek C.2: Mincovka.

Obrázek C.3: Foto testovací sady.